Compressive Sensing

Dr.-Ing. Ali Bereyhi

Friedrich-Alexander University

ali.bereyhi@fau.de

Summer 2022

Where are We?

- *Optimal recovery is given by ℓ_0-norm minimization*
  - *We can perform perfect recovery with at most 2s samples*
  - *It is however computationally infeasible*
- *Best relaxation of optimal approach is ℓ_1-norm minimization*
  - *We recover a signal with # of non-zero entries $\leq$ # of samples*
  - *We can feasibly implement it in various forms*
    - *Basis pursuit*
    - *Basis pursuit with quadratic constraint*
    - *Basis pursuit denoising*
    - *LASSO*

Where are We?

For some applications,

ℓ_1-norm minimization is still computationally complex

what we can do in such applications?

We could go for much lighter approaches

- *Greedy algorithms*
- *Thresholding algorithms*

We are going to learn them in this part

Iterative Sparse Recovery Algorithms

Back to the Original Problem

Sparse Recovery

Given sampling matrix $\mathbf{A} \in \mathbb{R}^{M \times N}$ *and samples* $\boldsymbol{y} \in \mathbb{R}^{M}$, *we want to find the sparse signal* $\boldsymbol{x} \in \mathbb{R}^{N}$ *from*

$$\mathbf{A}\boldsymbol{x} = \boldsymbol{y} \quad \textit{subject to } \|\boldsymbol{x}\|_0 \leq s$$

What are the solutions we have learned till today?

We find $\boldsymbol{x}$ *as*

$$\boldsymbol{x} = \operatorname{argmin} \|\boldsymbol{z}\|_p \quad \textit{subject to } \mathbf{A}\boldsymbol{z} = \boldsymbol{y}$$

for $p = 0$ *or* $p = 1$

ℓ_p-Norm Minimization vs Iterative Algorithms

How many steps does it take to do ℓ_p-norm minimization?

Just One!

- *We get the sampling matrix and samples*
- *We put them in the optimization problem*

However, this single step could be computationally complex!

An alternative approach is the iterative approach

- *We start with a bad approximation of the signal*
- *We improve it gradually by repeating a same operation*

Sparse Recovery via Iterative Algorithms

How do iterative algorithms perform sparse recovery?

They start from an initial choice $\boldsymbol{x}^{(0)}$ *and iterate as*

$$\boldsymbol{x}^{(t)} = F\left(\mathbf{A}, \boldsymbol{y}, \boldsymbol{x}^{(t-1)}\right)$$

They stop at iteration T *when* $\mathbf{A}\boldsymbol{x}^{(T)} = \boldsymbol{y}$

Sparse Recovery via Iterative Algorithms

Why should an iterative algorithm reduce the complexity?

We perform low-complexity operation in each iteration

Then, why don't we always use them?

Low-complexity comes at the cost of poor performance

We need more samples to recover $\boldsymbol{x}$ *by an iterative algorithm*

We now go through the important iterative algorithms

Greedy Algorithms

Greedy Algorithms

How does a greedy algorithm work?

- *It starts with* $\boldsymbol{x}^{(0)} = \mathbf{0}$ *and* $\mathcal{S}^{(0)} = \emptyset$
- *It updates the support for* $t \geq 1$ *as*

$$\mathcal{S}^{(t)} = \mathcal{S}^{(t-1)} \cup S^{+}\left(\boldsymbol{y}, \boldsymbol{x}^{(t-1)}\right) - S^{-}\left(\boldsymbol{y}, \boldsymbol{x}^{(t-1)}\right)$$

- *It finds best* $\boldsymbol{z}$ *for approximation* $\mathbf{A}_{\mathcal{S}^{(t)}} \boldsymbol{z} \approx \boldsymbol{y}$, *and set*

$$\boldsymbol{x}^{(t)}_{\mathcal{S}^{(t)}} = \boldsymbol{z}$$

- *It stops at iteration* T, *at which* $\mathbf{A}\boldsymbol{x}^{(T)} = \boldsymbol{y}$

Greedy Algorithms

Why should such an algorithm work?

Assume a genie tells us that the support of signal is $\mathcal{S}$

- *We consider a vector $\boldsymbol{z} \in \mathbb{R}^{|\mathcal{S}|}$ and solve*

$$\mathbf{A}_{\mathcal{S}} \boldsymbol{z} = \boldsymbol{y}$$

  which is a determined system of equations

- *The solution is then given by $\boldsymbol{x}$ such that*

$$\boldsymbol{x}_{\mathcal{S}} = \boldsymbol{z} \ \text{ and } \ \boldsymbol{x}_{\bar{\mathcal{S}}} = \mathbf{0}$$

Greedy Algorithms

Why should such an algorithm work?

The key problem is that

We do not have a genie!

We however know that

Whatever $\mathcal{S}$ is, it describes the sparsest possible set

So we start from $\mathcal{S}^{(0)} = \emptyset$ and gradually extend it till

We get to the sparsest possible set

Greedy Algorithms

Now, let us have a deeper look on a greedy algorithm

- *Start with* $\boldsymbol{x}^{(0)} = \mathbf{0}$ *and* $\mathcal{S}^{(0)} = \emptyset$
- *Update the support for* $t \geq 1$ *as*

$$\mathcal{S}^{(t)} = \mathcal{S}^{(t-1)} \cup S^{+}\left(\boldsymbol{y}, \boldsymbol{x}^{(t-1)}\right) - S^{-}\left(\boldsymbol{y}, \boldsymbol{x}^{(t-1)}\right)$$

- *Find best* $\boldsymbol{z}$ *for approximation* $\mathbf{A}_{\mathcal{S}^{(t)}}\boldsymbol{z} \approx \boldsymbol{y}$, *and set*

$$\boldsymbol{x}^{(t)}_{\mathcal{S}^{(t)}} = \boldsymbol{z}$$

- *Stop at iteration* T, *at which* $\mathbf{A}\boldsymbol{x}^{(T)} = \boldsymbol{y}$

Greedy Algorithms

Now, let us have a deeper look on a greedy algorithm

In iteration t, we could see

- $\mathcal{S}^{(t)}$ *as the estimation of support*
- $\boldsymbol{x}^{(t)}$ *as the estimation of the sparse signal*

So, we could say

The greedy algorithm . . .

- *starts with sparsest estimation of signal ever*
- *improves estimation gradually by extending the support*

Greedy Algorithms

The key computational tasks in a greedy algorithms are

- *Red Step: Add new indices or remove some to/from support*

$$S^{+}\left(\boldsymbol{y}, \boldsymbol{x}^{(t-1)}\right) \qquad \text{and} \qquad S^{-}\left(\boldsymbol{y}, \boldsymbol{x}^{(t-1)}\right)$$

  This function takes $\boldsymbol{y}$ *and* $\boldsymbol{x}^{(t-1)}$ *and gives new indices*

  - *There are several methods to realize* $S(\cdot)$

- *Blue Step: We find best estimation in iteration* t
  - *This is done by a unique approach*

Greedy Algorithms

We could hence say that

Key Indicator of a Greedy Algorithm

Different greedy sparse recovery algorithms differ in the design of

$$S^{+}\left(\boldsymbol{y}, \boldsymbol{x}^{(t-1)}\right) \qquad \text{and} \qquad S^{-}\left(\boldsymbol{y}, \boldsymbol{x}^{(t-1)}\right)$$

Before we go for the well-known greedy algorithms, let us explain

the universally common estimation used in the blue step

Greedy Algorithms: Least Squares Estimation

We now focus on the blue step

We are given with support $\mathcal{S}^{(t)}$ *which contains* s_t *indices*

We want to find best $\boldsymbol{z} \in \mathbb{R}^{s_t}$ *for approximation* $\mathbf{A}_{\mathcal{S}^{(t)}} \boldsymbol{z} \approx \boldsymbol{y}$

First of all, we know that

$$\mathbf{A}_{\mathcal{S}^{(t)}} \boldsymbol{z} = \boldsymbol{y} \text{ should be an overdetermined problem}$$

Otherwise, $\boldsymbol{z}$ *would have been a solution of* $\mathbf{A}_{\mathcal{S}^{(t)}} \boldsymbol{z} = \boldsymbol{y}$!

Greedy Algorithms: Least Squares Estimation

We now focus on the blue step

We are given with support $\mathcal{S}^{(t)}$ which contains s_t indices

We want to find best $\boldsymbol{z} \in \mathbb{R}^{s_t}$ for approximation $\mathbf{A}_{\mathcal{S}^{(t)}} \boldsymbol{z} \approx \boldsymbol{y}$

Since we deal with an overdetermined problem,

We have no solution for $\mathbf{A}_{\mathcal{S}^{(t)}} \boldsymbol{z} = \boldsymbol{y}$

So, we find that choice of $\boldsymbol{z}$ which

puts $\mathbf{A}_{\mathcal{S}^{(t)}} \boldsymbol{z}$ as close as possible to $\boldsymbol{y}$

Greedy Algorithms: Least Squares Estimation

We now focus on the blue step

We are given with support $\mathcal{S}^{(t)}$ *which contains* s_t *indices*

We want to find best $\boldsymbol{z} \in \mathbb{R}^{s_t}$ *for approximation* $\mathbf{A}_{\mathcal{S}^{(t)}} \boldsymbol{z} \approx \boldsymbol{y}$

$\mathbf{A}_{\mathcal{S}^{(t)}} \boldsymbol{z}$ *as close as possible to* $\boldsymbol{y}$ *means*

$$\boldsymbol{z} = \underset{\boldsymbol{v}}{\operatorname{argmin}} \|\mathbf{A}_{\mathcal{S}^{(t)}} \boldsymbol{v} - \boldsymbol{y}\|_2^2$$

This is known the method of least-squares or simply LS method

Greedy Algorithms: Least Squares Estimation

We now focus on the blue step

We are given with support $\mathcal{S}^{(t)}$; then, we set

$$\boldsymbol{z} = \underset{\boldsymbol{v}}{\operatorname{argmin}} \|\mathbf{A}_{\mathcal{S}^{(t)}} \boldsymbol{v} - \boldsymbol{y}\|_2^2$$

One might ask why is it called the method of least-squares?

Well! We find $\boldsymbol{z}$ such that the sum of squared errors is the least

Attention: Keep the LS method in mind, as we use it a lot!

Greedy Algorithms: Least Squares Estimation

How to solve the optimization in the LS method?

The objective function

$$f\left(\boldsymbol{v}\right) = \left\|\mathbf{A}_{\mathcal{S}^{(t)}}\boldsymbol{v} - \boldsymbol{y}\right\|_2^2$$

is convex. So, we calculate the gradient

$$\nabla f\left(\boldsymbol{v}\right) = \begin{bmatrix} \dfrac{\partial}{\partial v_1} f\left(\boldsymbol{v}\right) \\ \vdots \\ \dfrac{\partial}{\partial v_{s_t}} f\left(\boldsymbol{v}\right) \end{bmatrix} = 2\mathbf{A}_{\mathcal{S}^{(t)}}^{\mathsf{T}} \left(\mathbf{A}_{\mathcal{S}^{(t)}}\boldsymbol{v} - \boldsymbol{y}\right)$$

Greedy Algorithms: Least Squares Estimation

How to solve the optimization in the LS method?

We then find $\mathbf{z}$

$$\nabla f(\mathbf{z}) = \mathbf{0} \rightsquigarrow 2\mathbf{A}_{\mathcal{S}(t)}^{\mathsf{T}} \left(\mathbf{A}_{\mathcal{S}(t)} \mathbf{z} - \mathbf{y} \right) = \mathbf{0}$$

This concludes

$$\mathbf{A}_{\mathcal{S}(t)}^{\mathsf{T}} \mathbf{A}_{\mathcal{S}(t)} \mathbf{z} = \mathbf{A}_{\mathcal{S}(t)}^{\mathsf{T}} \mathbf{y}$$

Or equivalently

$$\mathbf{z} = \left(\mathbf{A}_{\mathcal{S}(t)}^{\mathsf{T}} \mathbf{A}_{\mathcal{S}(t)} \right)^{-1} \mathbf{A}_{\mathcal{S}(t)}^{\mathsf{T}} \mathbf{y}$$

Greedy Algorithms: Least Squares Estimation

How to solve the optimization in the LS method?

We often write $\boldsymbol{z}$ *as*

$$\boldsymbol{z} = \mathbf{A}_{\mathcal{S}(t)}^{\dagger} \boldsymbol{y}$$

and define

$$\mathbf{A}_{\mathcal{S}(t)}^{\dagger} = \left(\mathbf{A}_{\mathcal{S}(t)}^{\mathsf{T}} \mathbf{A}_{\mathcal{S}(t)} \right)^{-1} \mathbf{A}_{\mathcal{S}(t)}^{\mathsf{T}}$$

as the pseudo-inverse of $\mathbf{A}_{\mathcal{S}(t)}$

Greedy Algorithms: Least Squares Estimation

Getting back to the blue step

We are given with support $\mathcal{S}^{(t)}$*; then, we set*

$$\boldsymbol{z} = \mathbf{A}^{\dagger}_{\mathcal{S}^{(t)}} \boldsymbol{y}$$

Once we have $\boldsymbol{z}$*, we set* $\boldsymbol{x}^{(t)}$ *as* $\boldsymbol{x}^{(t)}_{\mathcal{S}^{(t)}} = \boldsymbol{z}$ *and zero elsewhere*

We hence have the following property

$$\mathbf{A}\boldsymbol{x}^{(t)} = \mathbf{A}_{\mathcal{S}^{(t)}} \boldsymbol{x}^{(t)}_{\mathcal{S}^{(t)}} = \mathbf{A}_{\mathcal{S}^{(t)}} \boldsymbol{z}$$

Greedy Algorithms: Least Squares Estimation

Remember that $\boldsymbol{z}$ *was the solution of*

$$\mathbf{A}_{\mathcal{S}^{(t)}}^{\mathsf{T}} \left(\mathbf{A}_{\mathcal{S}^{(t)}} \boldsymbol{z} - \boldsymbol{y} \right) = \mathbf{0}$$

Since $\mathbf{A}\boldsymbol{x}^{(t)} = \mathbf{A}_{\mathcal{S}^{(t)}} \boldsymbol{z}$, *we could say*

$$\left(\mathbf{A}^{\mathsf{T}} \left(\mathbf{A}\boldsymbol{x}^{(t)} - \boldsymbol{y} \right) \right)_{\mathcal{S}^{(t)}} = \mathbf{0}$$

Let us now define the residual in iteration t *as*

$$\boldsymbol{r}^{(t)} = \mathbf{A}\boldsymbol{x}^{(t)} - \boldsymbol{y}$$

$\boldsymbol{r}^{(t)}$ *determines the estimation error in iteration* t

Greedy Algorithms: Orthogonality Principle

The residual in iteration t is

$$\boldsymbol{r}^{(t)} = \mathbf{A}\boldsymbol{x}^{(t)} - \boldsymbol{y}$$

Using this definition, we could say in iteration t, we have

$$\left(\mathbf{A}^{\mathsf{T}}\boldsymbol{r}^{(t)}\right)_{\mathcal{S}^{(t)}} = \mathbf{0}$$

This is in fact a general property of the LS method which is called

Orthogonality Principle

Greedy Algorithms: Orthogonality Principle

The residual in iteration t is

$$\boldsymbol{r}^{(t)} = \mathbf{A}\boldsymbol{x}^{(t)} - \boldsymbol{y}$$

Orthogonality Principle

In iteration t of a greedy algorithm, sampling matrix is orthogonal to the residual on the estimated support, i.e.,

$$\left(\mathbf{A}^{\mathsf{T}}\boldsymbol{r}^{(t)}\right)_{\mathcal{S}^{(t)}} = \mathbf{0}$$

Greedy Algorithms

Back to the generic form of a greedy algorithm

- *Start with* $\boldsymbol{x}^{(0)} = \mathbf{0}$ *and* $\mathcal{S}^{(0)} = \emptyset$
- *Update the support for* $t \geq 1$ *as*

$$\mathcal{S}^{(t)} = \mathcal{S}^{(t-1)} \cup S^{+}\left(\boldsymbol{y}, \boldsymbol{x}^{(t-1)}\right) - S^{-}\left(\boldsymbol{y}, \boldsymbol{x}^{(t-1)}\right)$$

- *Set* $\boldsymbol{z} = \mathbf{A}^{\dagger}_{\mathcal{S}^{(t)}} \boldsymbol{y}$ *and update* $\boldsymbol{x}^{(t)}$ *as*

$$\boldsymbol{x}^{(t)}_{\mathcal{S}^{(t)}} = \boldsymbol{z}$$

- *Stop at iteration* T, *at which* $\mathbf{A}\boldsymbol{x}^{(T)} = \boldsymbol{y}$

Greedy Algorithms

Now, we focus on the red step

We are given with $\mathbf{x}^{(t-1)}$ whose support is $\mathcal{S}^{(t-1)}$ and want to add/remove indices to/from $\mathcal{S}^{(t-1)}$, such that $\mathbf{x}^{(t)}$ gets better

In the generic algorithm this task is done by the functions

$$S^{+}\left(\mathbf{y}, \mathbf{x}^{(t-1)}\right) \qquad \text{and} \qquad S^{-}\left(\mathbf{y}, \mathbf{x}^{(t-1)}\right)$$

These functions in general can be implemented in various forms

We discuss some known forms in the sequel

Greedy Algorithms

Section 1: Orthogonal Matching Pursuit

Orthogonal Matching Pursuit

Orthogonal matching pursuit is abbreviated as OMP

OMP adds only one index in each iteration, i.e., in iteration t

$$\{1, \dots, N\} - \mathcal{S}^{(t-1)} \ni n^{(t)} = S^{+}\left(\boldsymbol{y}, \boldsymbol{x}^{(t-1)}\right)$$

is added to the support

How does OMP find $n^{(t)}$?

It tries an step-wise approach

It finds $n^{(t)}$, such that estimation error decreases maximally

Orthogonal Matching Pursuit

OMP finds $n^{(t)}$ such that estimation error maximally decreases

Assume that OMP selects index $n^{(t)}$ in iteration t; thus,

$$\mathcal{S}^{(t)} = \mathcal{S}^{(t-1)} \cup \left\{ n^{(t)} \right\}$$

This means that

$\boldsymbol{x}^{(t)}$ *has non-zeros exactly where $\boldsymbol{x}^{(t-1)}$ is non-zero + at index $n^{(t)}$*

Orthogonal Matching Pursuit

OMP finds $n^{(t)}$ such that estimation error maximally decreases

Remember that $\boldsymbol{x}^{(t)}$ is determined via the LS method, i.e.,

Among all vectors whose supports $\subseteq \mathcal{S}^{(t)}$,

$\boldsymbol{x}^{(t)}$ gives minimum estimation error

In other words,

$$\|\boldsymbol{y} - \mathbf{A}\boldsymbol{x}^{(t)}\|_2^2 \leq \|\boldsymbol{y} - \mathbf{A}\boldsymbol{v}\|_2^2 \qquad \forall \boldsymbol{v} : \mathrm{Supp}\left(\boldsymbol{v}\right) \subseteq \mathcal{S}^{(t)}$$

Orthogonal Matching Pursuit

OMP finds $n^{(t)}$ such that estimation error maximally decreases

Now, we construct $\boldsymbol{v}$ as below

$$\boldsymbol{v} = \boldsymbol{x}^{(t-1)} + \alpha\, \boldsymbol{e}_{n^{(t)}}$$

where $\boldsymbol{e}_{n^{(t)}}$ has only one entry 1 at index $n^{(t)}$

Since $\mathrm{Supp}\,(\boldsymbol{v}) \subseteq \mathcal{S}^{(t)}$, *we have*

$$\|\boldsymbol{y} - \mathbf{A}\boldsymbol{x}^{(t)}\|_2^2 \leq \|\boldsymbol{y} - \mathbf{A}\boldsymbol{v}\|_2^2$$

Orthogonal Matching Pursuit

OMP finds $n^{(t)}$ such that estimation error maximally decreases

So, we expand the right hand side

$$\begin{aligned}\|\boldsymbol{r}^{(t)}\|_2^2 = \|\boldsymbol{y} - \mathbf{A}\boldsymbol{x}^{(t)}\|_2^2 &\leq \|\boldsymbol{y} - \mathbf{A}\boldsymbol{v}\|_2^2 \\ &= \|\boldsymbol{y} - \mathbf{A}\left(\boldsymbol{x}^{(t-1)} + \alpha \boldsymbol{e}_{n^{(t)}}\right)\|_2^2 \\ &= \|\boldsymbol{y} - \mathbf{A}\boldsymbol{x}^{(t-1)} - \alpha \mathbf{A}\boldsymbol{e}_{n^{(t)}}\|_2^2 \\ &= \|\boldsymbol{r}^{(t-1)} - \alpha \mathbf{A}\boldsymbol{e}_{n^{(t)}}\|_2^2\end{aligned}$$

Orthogonal Matching Pursuit

OMP finds $n^{(t)}$ such that estimation error maximally decreases

Keep in mind that for any $\boldsymbol{z} \in \mathbb{R}^N$,

$$\boxed{\boldsymbol{z}^\mathsf{T} \boldsymbol{e}_{n^{(t)}} \text{ selects } z_{n^{(t)}}}$$

We hence can write

$$\begin{aligned} \|\boldsymbol{r}^{(t)}\|_2^2 &\leq \|\boldsymbol{r}^{(t-1)} - \alpha \mathbf{A} \boldsymbol{e}_{n^{(t)}}\|_2^2 \\ &= \|\boldsymbol{r}^{(t-1)}\|_2^2 + \alpha^2 \|\boldsymbol{a}_{n^{(t)}}\|_2^2 - 2\alpha \left[\mathbf{A}^\mathsf{T} \boldsymbol{r}^{(t-1)}\right]_{n^{(t)}} \end{aligned}$$

where $\boldsymbol{a}_{n^{(t)}}$ the $n^{(t)}$-th column of $\mathbf{A}$

Orthogonal Matching Pursuit

OMP finds $n^{(t)}$ such that estimation error maximally decreases

Often the columns of $\mathbf{A}$ are normalized, i.e.,

$$\|\boldsymbol{a}_{n^{(t)}}\|_2^2 = 1$$

We hence can write

$$\|\boldsymbol{r}^{(t)}\|_2^2 \leq \|\boldsymbol{r}^{(t-1)}\|_2^2 + \alpha^2 - 2\alpha \left[\mathbf{A}^\mathsf{T} \boldsymbol{r}^{(t-1)}\right]_{n^{(t)}}$$

Or equivalently

$$2\alpha \left[\mathbf{A}^\mathsf{T} \boldsymbol{r}^{(t-1)}\right]_{n^{(t)}} - \alpha^2 \leq \|\boldsymbol{r}^{(t-1)}\|_2^2 - \|\boldsymbol{r}^{(t)}\|_2^2 = \Delta_{n^{(t)}}$$

Orthogonal Matching Pursuit

OMP finds $n^{(t)}$ such that estimation error maximally decreases

So, we could conclude that in each iteration

$$\|\boldsymbol{r}^{(t)}\|_2^2 = \|\boldsymbol{r}^{(t-1)}\|_2^2 - \Delta_{n^{(t)}}$$

where $\Delta_{n^{(t)}}$ is bounded from below as

$$\Delta_{n^{(t)}} \geq 2\alpha \left[\mathbf{A}^\mathsf{T} \boldsymbol{r}^{(t-1)}\right]_{n^{(t)}} - \alpha^2$$

Orthogonal Matching Pursuit

OMP finds $n^{(t)}$ such that estimation error maximally decreases

To make the bound as tight as possible, we consider its maximum

$$\begin{aligned}\Delta_{n^{(t)}} &\geq \max_{\alpha} 2\alpha \left[\mathbf{A}^{\mathsf{T}} \boldsymbol{r}^{(t-1)}\right]_{n^{(t)}} - \alpha^2 \\ &= \left| \left[\mathbf{A}^{\mathsf{T}} \boldsymbol{r}^{(t-1)}\right]_{n^{(t)}} \right|^2\end{aligned}$$

So we finally have the following lower bound on $\Delta_{n^{(t)}}$

$$\Delta_{n^{(t)}} \geq \left| \left[\mathbf{A}^{\mathsf{T}} \boldsymbol{r}^{(t-1)}\right]_{n^{(t)}} \right|^2$$

Orthogonal Matching Pursuit

OMP finds $n^{(t)}$ such that estimation error maximally decreases

Since OMP cannot find the exact estimation error,

OMP finds $n^{(t)}$ such that lower bound maximally decreases

So OMP selects $n^{(t)}$ as follows:

$$
\begin{aligned}
n^{(t)} = S\left(\boldsymbol{y}, \boldsymbol{x}^{(t-1)}\right) &= \underset{n}{\operatorname{argmax}} \left| \left[\mathbf{A}^{\mathsf{T}} \boldsymbol{r}^{(t-1)} \right]_n \right| \\
&= \underset{n}{\operatorname{argmax}} \left| \left[\mathbf{A}^{\mathsf{T}} \left(\boldsymbol{y} - \mathbf{A}\boldsymbol{x}^{(t-1)} \right) \right]_n \right|
\end{aligned}
$$

Orthogonal Matching Pursuit

Should we search over $\{1, \dots, N\} - \mathcal{S}^{(t-1)}$*?*

Interestingly No! Orthogonality principle guarantees that

$$n^{(t)} \notin \mathcal{S}^{(t-1)}$$

Orthogonality principle indicates that in iteration t*, we have*

$$\left(\mathbf{A}^{\mathsf{T}} \boldsymbol{r}^{(t)} \right)_{\mathcal{S}^{(t)}} = \mathbf{0}$$

which means that for any $j \in \mathcal{S}^{(t-1)}$*, we have*

$$0 = \left| \left[\mathbf{A}^{\mathsf{T}} \boldsymbol{r}^{(t-1)} \right]_j \right| \leq \max_n \left| \left[\mathbf{A}^{\mathsf{T}} \boldsymbol{r}^{(t-1)} \right]_n \right|$$

Orthogonal Matching Pursuit

- *Start with* $\boldsymbol{x}^{(0)} = \mathbf{0}$ *and* $\mathcal{S}^{(0)} = \emptyset$
- *Update the support for* $t \geq 1$ *as*

$$\mathcal{S}^{(t)} = \mathcal{S}^{(t-1)} \cup \operatorname*{argmax}_n \left| \left[\mathbf{A}^{\mathsf{T}} \left(\boldsymbol{y} - \mathbf{A}\boldsymbol{x}^{(t-1)} \right) \right]_n \right|$$

- *Set* $\boldsymbol{z} = \mathbf{A}^{\dagger}_{\mathcal{S}^{(t)}} \boldsymbol{y}$ *and update* $\boldsymbol{x}^{(t)}$ *as*

$$\boldsymbol{x}^{(t)}_{\mathcal{S}^{(t)}} = \boldsymbol{z}$$

- *Stop at iteration* T, *at which* $\mathbf{A}\boldsymbol{x}^{(T)} = \boldsymbol{y}$

Orthogonal Matching Pursuit

Now assume that we have sampled an

$$s\text{-sparse signal } \boldsymbol{x}^\star \in \mathbb{R}^N$$

How many iterations does OMP need to recover $\boldsymbol{x}^\star \in \mathbb{R}^N$*?*

If it converges, it should do it in s *iterations*

Is there any guarantee that this happens?

If $\mathbf{A}$ *has some good properties, Yes!*

We see a basic result in the sequel

Greedy Algorithms

Section 2: *Compressive Sampling Matching Pursuit*

Main Drawbacks of OMP

The key drawbacks of OMP are ...

- *It adds only one index to the support per iteration*
- *It always adds indices*

> *Once a wrong index is added,*
>
> *It always remains in the recovered support!*
>
> *Let's call it support detection issue*

How could we get rid of the support detection issue?

Compressive Sampling Matching Pursuit (CoSaMP)

CoSaMP: Derivation

Back to the generic form of a greedy algorithm

- *Start with* $\boldsymbol{x}^{(0)} = \mathbf{0}$ *and* $\mathcal{S}^{(0)} = \emptyset$
- *Update the support for* $t \geq 1$ *as*

$$\mathcal{S}^{(t)} = \mathcal{S}^{(t-1)} \cup S\left(\boldsymbol{y}, \boldsymbol{x}^{(t-1)}\right)$$

- *Set* $\boldsymbol{z} = \mathbf{A}^{\dagger}_{\mathcal{S}^{(t)}} \boldsymbol{y}$ *and update* $\boldsymbol{x}^{(t)}$ *as*

$$\boldsymbol{x}^{(t)}_{\mathcal{S}^{(t)}} = \boldsymbol{z}$$

- *Stop at iteration* T*, at which* $\mathbf{A}\boldsymbol{x}^{(T)} = \boldsymbol{y}$

CoSaMP: Derivation

The key difference of CoSaMP to the OMP is the fact that

CoSaMP adds and removes multiple indices in each iteration

How does CoSaMP do this?

We add the K indices in $\mathcal{I}^{(t)} = \{n_1, \ldots, n_K\}$ in iteration t

Estimation error now drops with $\Delta^{(t)} = \|\boldsymbol{r}^{(t-1)}\|_2^2 - \|\boldsymbol{r}^{(t)}\|_2^2$

Similar to OMP, we could show that

$$\Delta^{(t)} \propto \left\| \left[\mathbf{A}^{\mathsf{T}} \boldsymbol{r}^{(t-1)} \right]_{\mathcal{I}^{(t)}} \right\|^2$$

CoSaMP: Derivation

What is the conclusion?

> *We could still choose those indices with largest*
>
> $$\left| \left[\mathbf{A}^{\mathsf{T}} \boldsymbol{r}^{(t-1)} \right]_n \right|^2$$
>
> *This means that we are still doing Matching Pursuit*

Let us define dominant support selector which does this

$$D_s\left(\boldsymbol{x}\right) = \{\textit{Indices of } s \textit{ entries of } \boldsymbol{x} \textit{ with largest absolute value}\}$$

CoSaMP: Derivation

What is the conclusion?

> *We could still choose those indices with largest*
>
> $$\left| \left[\mathbf{A}^{\mathsf{T}} \boldsymbol{r}^{(t-1)} \right]_n \right|^2$$
>
> *This means that we are still doing Matching Pursuit*

So we could say

$$\mathcal{I}^{(t)} = D_K \left(\mathbf{A}^{\mathsf{T}} \left(\boldsymbol{y} - \mathbf{A} \boldsymbol{x}^{(t-1)} \right) \right)$$

CoSaMP: Derivation

What happens when we add multiple indices?

We already have the support detection issue in OMP

So, we could conclude that

The issue gets more severe as we increase the step size K

How does CoSaMP deal with this issue? In each iteration,

- *It first adds multiple indices via matching pursuit*
- *It then modifies $\mathcal{S}^{(t)}$ by removing less-effective indices*

CoSaMP: Derivation

How does CoSaMP select the less-effective indices?

Let us update the support in iteration t such that

$$\left|\mathcal{S}^{(t)}\right| \geq s$$

Now we apply the LS method and find a signal $\boldsymbol{u}^{(t)}$

$$\boldsymbol{u}^{(t)} \text{ has more than } s \text{ non-zero entries}$$

Clearly, entries of $\boldsymbol{u}^{(t)}$ which have smaller absolute value are

less-effective in the approximation of the samples

CoSaMP: Derivation

How does CoSaMP update the support?

CoSaMP updates $\mathcal{S}^{(t)}$ once again in iteration t to be

$$\mathcal{S}^{(t)} = D_s\left(\boldsymbol{u}^{(t)}\right)$$

Clearly, it also needs to update the estimation by

getting rid of less-effective entries, i.e.,

$$x_n^{(t)} = \begin{cases} u_n^{(t)} & \forall n \in D_s\left(\boldsymbol{u}^{(t)}\right) \\ 0 & \forall n \notin D_s\left(\boldsymbol{u}^{(t)}\right) \end{cases}$$

CoSaMP: Derivation

Let us define the hard thresholding operator

$$T_s^{\mathrm{H}}(\boldsymbol{x}) = \begin{cases} x_n & n \in D_s(\boldsymbol{x}) \\ 0 & n \notin D_s(\boldsymbol{x}) \end{cases}$$

By hard thresholding, we approximate the signal with

Its most effective s-subvector

and set the remaining entries zero

CoSaMP: Derivation

How does CoSaMP update the support?

CoSaMP updates $\mathcal{S}^{(t)}$ once again in iteration t to be

$$\mathcal{S}^{(t)} = D_s\left(\boldsymbol{u}^{(t)}\right)$$

Then, it updates the estimation as

$$\boldsymbol{x}^{(t)} = T_s^{\mathrm{H}}\left(\boldsymbol{u}^{(t)}\right)$$

CoSaMP Algorithm

- *Start with* $\mathbf{x}^{(0)} = \mathbf{0}$ *and* $\mathcal{S}^{(0)} = \emptyset$, *and choose a* K
- *Update the support for* $t \geq 1$ *as*

$$\hat{\mathcal{S}}^{(t)} = \mathcal{S}^{(t-1)} \cup D_K \left(\mathbf{A}^\mathsf{T} \left(\mathbf{y} - \mathbf{A}\mathbf{x}^{(t-1)} \right) \right)$$

- *Set* $\mathbf{z} = \mathbf{A}^\dagger_{\hat{\mathcal{S}}^{(t)}} \mathbf{y}$ *and update* $\mathbf{u}^{(t)}$ *as* $\mathbf{u}^{(t)}_{\hat{\mathcal{S}}^{(t)}} = \mathbf{z}$
- *Update once again* $\mathcal{S}^{(t)} = D_s\left(\mathbf{u}^{(t)}\right)$ *and* $\mathbf{x}^{(t)} = T_s^{\mathrm{H}}\left(\mathbf{u}^{(t)}\right)$
- *Stop at iteration* T, *at which* $\mathbf{A}\mathbf{x}^{(T)} = \mathbf{y}$

CoSaMP Algorithm

What is the typical choice for K?

We usually set $K = 2s$

What is the reason for such a choice?

The exact reasoning needs some derivations; however, we know

It has connections to the bound $M \geq 2s$ for optimal recovery

CoSaMP vs OMP

What are the pros of CoSaMP compared to OMP?

- *It updates the support with a larger step size*
- *It addresses the support detection issue*

So, if CoSaMP is so good, why should we still study OMP?

CoSaMP needs to know the sparsity s in advance!

- *OMP does the sparse recovery blindly*
- *CoSaMP does the sparse recovery based on s*

If we are wrong about s; then, CoSaMP can perform poorly!

Greedy Algorithms

Section 3: *Subspace Pursuit*

Orthogonality Principle and CoSaMP

Let's look back to the CoSaMP algorithm

- *Start with* $\boldsymbol{x}^{(0)} = \mathbf{0}$ *and* $\mathcal{S}^{(0)} = \emptyset$, *and choose a* K
- *Update the support for* $t \geq 1$ *as*

$$\hat{\mathcal{S}}^{(t)} = \mathcal{S}^{(t-1)} \cup D_K \left(\mathbf{A}^{\mathsf{T}} \left(\boldsymbol{y} - \mathbf{A}\boldsymbol{x}^{(t-1)} \right) \right)$$

- *Set* $\boldsymbol{z} = \mathbf{A}^{\dagger}_{\hat{\mathcal{S}}^{(t)}} \boldsymbol{y}$ *and update* $\boldsymbol{u}^{(t)}$ *as* $\boldsymbol{u}^{(t)}_{\hat{\mathcal{S}}^{(t)}} = \boldsymbol{z}$
- *Update once again* $\mathcal{S}^{(t)} = D_s\left(\boldsymbol{u}^{(t)}\right)$ *and* $\boldsymbol{x}^{(t)} = T^{\mathrm{H}}_s\left(\boldsymbol{u}^{(t)}\right)$
- *Stop at iteration* T, *at which* $\mathbf{A}\boldsymbol{x}^{(T)} = \boldsymbol{y}$

Orthogonality Principle and CoSaMP

Does $\boldsymbol{u}^{(t)}$ fulfills the orthogonality principle?

Yes! $\boldsymbol{u}^{(t)}$ is determined via the LS method

Orthogonality Principle

In iteration t of a greedy algorithm, sampling matrix is orthogonal to the residual on the estimated support, i.e.,

$$\left(\mathbf{A}^{\mathsf{T}}\left(\mathbf{A}\boldsymbol{u}^{(t)} - \boldsymbol{y}\right)\right)_{\hat{\mathcal{S}}^{(t)}} = \mathbf{0}$$

Orthogonality Principle and CoSaMP

What about $\mathbf{x}^{(t)}$*?*

Well, we could check it! We know that $\mathcal{S}^{(t)} \subset \hat{\mathcal{S}}^{(t)}$*, so*

$$\left(\mathbf{A}^{\mathsf{T}} \left(\mathbf{A} \boldsymbol{u}^{(t)} - \boldsymbol{y} \right) \right)_{\mathcal{S}^{(t)}} = \mathbf{0}$$

Now, let us define $\boldsymbol{v}^{(t)} = \boldsymbol{u}^{(t)} - \boldsymbol{x}^{(t)}$*. We could hence write*

$$\left(\mathbf{A}^{\mathsf{T}} \left(\mathbf{A} \left(\boldsymbol{v}^{(t)} + \boldsymbol{x}^{(t)} \right) - \boldsymbol{y} \right) \right)_{\mathcal{S}^{(t)}} = \mathbf{0}$$

$$\left(\mathbf{A}^{\mathsf{T}} \mathbf{A} \boldsymbol{v}^{(t)} \right)_{\mathcal{S}^{(t)}} + \left(\mathbf{A}^{\mathsf{T}} \left(\mathbf{A} \boldsymbol{x}^{(t)} - \boldsymbol{y} \right) \right)_{\mathcal{S}^{(t)}} = \mathbf{0}$$

Orthogonality Principle and CoSaMP

What about $\mathbf{x}^{(t)}$?

We could hence write

$$\left(\mathbf{A}^{\mathsf{T}}\left(\mathbf{A}\mathbf{x}^{(t)} - \mathbf{y}\right)\right)_{\mathcal{S}^{(t)}} = -\left(\mathbf{A}^{\mathsf{T}}\mathbf{A}\mathbf{v}^{(t)}\right)_{\mathcal{S}^{(t)}}$$

which is not necessarily zero! Thus, the answer is No!

Subspace pursuit keeps the residual orthogonal

It once again performs the LS method on the updated support

- *Set $\mathbf{z} = \mathbf{A}^{\dagger}_{\mathcal{S}^{(t)}}\mathbf{y}$ and update $\mathbf{x}^{(t)}$ as $\mathbf{x}^{(t)}_{\mathcal{S}^{(t)}} = \mathbf{z}$*

Subspace Pursuit

- *Start with* $\boldsymbol{x}^{(0)} = \mathbf{0}$ *and* $\mathcal{S}^{(0)} = \emptyset$*, and choose a* K
- *Update initially the support for* $t \geq 1$ *as*

$$\hat{\mathcal{S}}^{(t)} = \mathcal{S}^{(t-1)} \cup D_K\left(\mathbf{A}^\mathsf{T}\left(\boldsymbol{y} - \mathbf{A}\boldsymbol{x}^{(t-1)}\right)\right)$$

- *Set* $\boldsymbol{z} = \mathbf{A}^\dagger_{\hat{\mathcal{S}}^{(t)}}\boldsymbol{y}$ *and update* $\boldsymbol{u}^{(t)}$ *as* $\boldsymbol{u}^{(t)}_{\hat{\mathcal{S}}^{(t)}} = \boldsymbol{z}$
- *Update* $\mathcal{S}^{(t)} = D_s\left(\boldsymbol{u}^{(t)}\right)$
- *Set* $\boldsymbol{z} = \mathbf{A}^\dagger_{\mathcal{S}^{(t)}}\boldsymbol{y}$ *and update* $\boldsymbol{x}^{(t)}$ *as* $\boldsymbol{x}^{(t)}_{\mathcal{S}^{(t)}} = \boldsymbol{z}$
- *Stop at iteration* T*, at which* $\mathbf{A}\boldsymbol{x}^{(T)} = \boldsymbol{y}$

CoSaMP vs OMP

What is the typical choice for K?

> *Typically, we set $K = s$*

What are the pros of subspace pursuit compared to CoSaMP?

> *It keeps the residual orthogonal*
> - *Matching pursuit updates the support more effectively*

Is there any cons?

> - *Subspace pursuit does the sparse recovery based on s*
> - *Higher computation, as it performs LS for two times*

Summary

We have learned greedy algorithms up to now

- *They update the support iterativel*y
- *In each iteration, they approximate signal via the LS method*
- *Various approaches for support selection*
  - *Orthogonal Matching Pursuit*
  - *Compressive Sampling Matching Pursuit*
  - *Subspace Pursuit*

We now start with thresholding algorithms

Thresholding Algorithms

Thresholding Algorithms

How does a thresholding algorithm work?

- *It starts with* $\boldsymbol{x}^{(0)} = \mathbf{0}$ *and* $\mathcal{S}^{(0)} = \emptyset$
- *It updates the approximation of the signal as*

$$\boldsymbol{x}^{(t)} = F\left(\boldsymbol{x}^{(t-1)}, \boldsymbol{y}, \mathbf{A}\right)$$

  for some thresholding function $F(\cdot)$
- *It stops at iteration* T, *at which* $\mathbf{A}\boldsymbol{x}^{(T)} = \boldsymbol{y}$

Thresholding Algorithms

What does a thresholding function do?

The thresholding function $F(\cdot)$ determines a sparse vector

- *It first calculates a vector $\boldsymbol{u}^{(t)}$ from $\mathbf{A}$, $\boldsymbol{y}$ and $\boldsymbol{x}^{(t)}$*
- *It then set some entries of $\boldsymbol{x}^{(t)}$ zero via thresholding*

What is the key characteristic of these algorithms?

They are computationally very cheap

Thresholding Algorithms

Why should such an algorithm work?

> *Simple example: We want to find solution of the equation*
>
> $$(x-1)^2 = 0$$

If we want to do it via a simple computer program, we could write

$$x^2 + 1 - 2x = 0 \rightsquigarrow x = \frac{x^2+1}{2}$$

This recursive equation describes a simple iterative approach

Thresholding Algorithms

Why should such an algorithm work?

> *Simple example: We want to find solution of the equation*
>
> $$(x-1)^2 = 0$$

We *iterate* *as below*

$$x^{(t)} = \frac{\left|x^{(t-1)}\right|^2 + 1}{2}$$

One can see

> *If we start with* $\left|x^{(0)}\right| \leq 1$, *we* *converge* *to solution* $x^{(\infty)} = 1$

Thresholding Algorithms

Why should such an algorithm work?

Another example: We want to find complex z

$$\left| z - e^{j\theta} \right|^2 = 0$$

and we know that the solution is on the unit circle, i.e., $|z| = 1$

We can again write

$$\left| z - e^{j\theta} \right|^2 = 0 \rightsquigarrow |z|^2 + 1 - e^{-j\theta} z - e^{j\theta} z^* = 0$$

Thus, we have the recursive equation

$$z = e^{j\theta} \left(|z|^2 + 1 \right) - e^{j2\theta} z^*$$

Thresholding Algorithms

Why should such an algorithm work?

Another example: We want to find complex z

$$\left| z - e^{j\theta} \right|^2 = 0$$

and we know that the solution is on the unit circle, i.e., $|z| = 1$

One would then suggest to iterate as

$$z^{(t)} = e^{j\theta} \left(\left| z^{(t-1)} \right|^2 + 1 \right) - e^{j2\theta} z^{(t-1)*}$$

However starting from $z^{(0)} \neq 0$, the algorithm diverges!

Thresholding Algorithms

Why should such an algorithm work?

Another example: We want to find complex z

$$\left| z - e^{j\theta} \right|^2 = 0$$

and we know that the solution is on the unit circle, i.e., $|z| = 1$

We could use the fact that $|z| = 1$ and modify the iteration as

$$u^{(t)} = e^{j\theta} \left(\left| z^{(t-1)} \right|^2 + 1 \right) - e^{j2\theta} z^{(t-1)^*}$$

$$z^{(t)} = \frac{u^{(t)}}{\left| u^{(t)} \right|}$$

Thresholding Algorithms

Why should such an algorithm work?

Another example: We want to find complex z

$$\left| z - e^{j\theta} \right|^2 = 0$$

and we know that the solution is on the unit circle, i.e., $|z| = 1$

Starting from any $z^{(0)}$ with $\left| z^{(0)} \right| = 1$

Now the algorithm converges to $z^{(\infty)} = e^{j\theta}$

Thresholding Algorithms

This is how the thresholding algorithms work

They start with a feasible sparse point

- *They iterate via a recursive equation*
- *They apply thresholding to keep the updated point sparse*

If the sensing matrix has good properties; then,

A thresholding algorithm can also lead us to the solution!

Thresholding Algorithms

Section 1: Iterative Hard Thresholding

Iterative Hard Thresholding: Derivation

What is the recursive equation here?

Let $\boldsymbol{x}$ *be the s-sparse solution*

$$\boldsymbol{x} - \boldsymbol{x} = \mathbf{A}^{\mathsf{T}}\mathbf{A}\left(\boldsymbol{x} - \boldsymbol{x}\right) = \mathbf{0}$$

Since $\boldsymbol{y} = \mathbf{A}\boldsymbol{x}$

$$\boldsymbol{x} - \boldsymbol{x} = \mathbf{A}^{\mathsf{T}}\left(\boldsymbol{y} - \mathbf{A}\boldsymbol{x}\right)$$

So, we can represent $\boldsymbol{x}$ *recursively as*

$$\boldsymbol{x} = \boldsymbol{x} + \mathbf{A}^{\mathsf{T}}\left(\boldsymbol{y} - \mathbf{A}\boldsymbol{x}\right)$$

Iterative Hard Thresholding: Derivation

We should also take into account the sparsity

Since the signal is s-sparse, we have

$$\boldsymbol{x} = T_s^{\mathrm{H}}(\boldsymbol{x})$$

Therefore we could write

$$\boldsymbol{x} = T_s^{\mathrm{H}}\left(\boldsymbol{x} + \mathbf{A}^{\mathsf{T}}(\boldsymbol{y} - \mathbf{A}\boldsymbol{x})\right)$$

Iterative Hard Thresholding

- *It starts with* $\boldsymbol{x}^{(0)} = \mathbf{0}$ *and* $\mathcal{S}^{(0)} = \emptyset$
- *It updates the approximation of the signal as*

$$\boldsymbol{x}^{(t)} = T_s^{\mathrm{H}} \left(\boldsymbol{x}^{(t-1)} + \mathbf{A}^{\mathsf{T}} \left(\boldsymbol{y} - \mathbf{A}\boldsymbol{x}^{(t-1)} \right) \right)$$

- *It stops at iteration* T*, at which* $\mathbf{A}\boldsymbol{x}^{(T)} = \boldsymbol{y}$

One could immediately see that this algorithm also

Performs sparse recovery based on s

So, it suffers from the same issue as CoSaMP and subspace pursuit

Iterative Hard Thresholding

How complicated is iterative hard thresholding?

It is one of the most low-complexity algorithms!

What about the performance?

It has a relatively poor performance!

Can we shift this performance-complexity trade-off?

We could try better estimation in each iteration by LS Method

Doing so, we end up with Hard Thresholding Pursuit

Thresholding Algorithms

Section 2: Hard Thresholding Pursuit

Hard Thresholding Pursuit: Derivation

In this scheme, we use hard thresholding

only for support recovery!

and perform the estimation in each iteration via the LS method

Estimation via LS

Given the estimated support $\mathcal{S}^{(t)}$ *in iteration* t*, we find*

$$\boldsymbol{z} = \mathbf{A}_{\mathcal{S}^{(t)}}^{\dagger} \boldsymbol{y}$$

$\boldsymbol{x}_{\mathcal{S}^{(t)}}^{(t)} = \boldsymbol{z}$*, and the remaining entries of* $\boldsymbol{x}^{(t)}$ *are set to zero*

Hard Thresholding Pursuit

- *It starts with* $\boldsymbol{x}^{(0)} = \mathbf{0}$ *and* $\mathcal{S}^{(0)} = \emptyset$
- *It updates the approximation of the support as*

$$\mathcal{S}^{(t)} = D_s \left(\boldsymbol{x}^{(t-1)} + \mathbf{A}^\mathsf{T} \left(\boldsymbol{y} - \mathbf{A}\boldsymbol{x}^{(t-1)} \right) \right)$$

- *It sets*

$$\boldsymbol{z} = \mathbf{A}^\dagger_{\mathcal{S}^{(t)}} \boldsymbol{y}$$

Updates $\boldsymbol{x}^{(t)}_{\mathcal{S}^{(t)}} = \boldsymbol{z}$ *and sets other entries to zero*

- *It stops at iteration* T*, at which* $\mathbf{A}\boldsymbol{x}^{(T)} = \boldsymbol{y}$

Hard Thresholding Pursuit

How is the complexity compared to iterative hard thresholding?

It has clearly higher complexity, as it needs to perform LS

What about the the performance?

It performs relatively better

One further observes that this algorithm also

Performs sparse recovery based on s

Thresholding Algorithms

Section 3: Iterative Soft Thresholding

Iterative Soft Thresholding: Derivation

Remember the definition of the residual

Let $\boldsymbol{z}$ be an estimation of the solution; then,

$$\boldsymbol{r}(\boldsymbol{z}) = \boldsymbol{y} - \mathbf{A}\boldsymbol{z}$$

Let us now define vector $\boldsymbol{u}(\boldsymbol{z})$ for some scalar β as follows

$$\boldsymbol{u}(\boldsymbol{z}) = \boldsymbol{z} + \beta \mathbf{A}^{\mathsf{T}} \boldsymbol{r}(\boldsymbol{z})$$

Clearly, for any solution of $\boldsymbol{y} = \mathbf{A}\boldsymbol{x}$, we have $\boldsymbol{r} = 0$, and thus

$$\boldsymbol{u}(\boldsymbol{x}) = \boldsymbol{x}$$

Iterative Soft Thresholding: Derivation

One could hence rewrite the sparse recovery problem as

Sparse Recovery Problem

$$\boldsymbol{z} = \boldsymbol{u}(\boldsymbol{z}) \qquad \textit{subject to} \;\; \|\boldsymbol{z}\|_0 \leq s$$

As we learned before, the solution is given by

ℓ_0-Norm Minimization

$$\min \|\boldsymbol{z}\|_0 \qquad \textit{subject to} \;\; \boldsymbol{z} = \boldsymbol{u}(\boldsymbol{z})$$

Iterative Soft Thresholding: Derivation

To be able to solve this problem, we relax it by ℓ_1-norm

ℓ_1-Norm Minimization

$$\min \|\boldsymbol{z}\|_1 \qquad \textit{subject to} \;\; \boldsymbol{z} = \boldsymbol{u}(\boldsymbol{z})$$

We now write the equivalent form with denoising

ℓ_1-Norm Minimization

$$\min \lambda \|\boldsymbol{z}\|_1 + \frac{1}{2}\|\boldsymbol{z} - \boldsymbol{u}(\boldsymbol{z})\|_2^2$$

Iterative Soft Thresholding: Derivation

This end up with this solution that

$$\boldsymbol{x} = \lim_{\lambda \to 0} \operatorname{argmin} \lambda \|\boldsymbol{z}\|_1 + \frac{1}{2} \|\boldsymbol{z} - \boldsymbol{u}(\boldsymbol{z})\|_2^2$$

Remember that

Basis Pursuit Solution $= \lim_{\lambda \to 0}$ *Basis Pursuit Denoising Solution*

Let us at the moment forget about the limit; thus,

$$\boldsymbol{x} = \operatorname{argmin} \lambda \|\boldsymbol{z}\|_1 + \frac{1}{2} \|\boldsymbol{z} - \boldsymbol{u}(\boldsymbol{z})\|_2^2$$

Iterative Soft Thresholding: Derivation

$$\boldsymbol{x} = \operatorname{argmin} \lambda \|\boldsymbol{z}\|_1 + \frac{1}{2} \|\boldsymbol{z} - \boldsymbol{u}(\boldsymbol{z})\|_2^2$$

We now turn it into a recursive equation: In iteration t,

- *Replace $\boldsymbol{u}(\boldsymbol{z})$ with $\boldsymbol{u}\left(\boldsymbol{x}^{(t-1)}\right)$*
- *Find the solution and call it $\boldsymbol{x}^{(t)}$*

So, we end up with

$$\boldsymbol{x}^{(t)} = \operatorname{argmin} \lambda \|\boldsymbol{z}\|_1 + \frac{1}{2} \|\boldsymbol{z} - \boldsymbol{u}\left(\boldsymbol{x}^{(t-1)}\right)\|_2^2$$

Iterative Soft Thresholding: Derivation

Let us now denote $\boldsymbol{u}^{(t)} = \boldsymbol{u}\left(\boldsymbol{x}^{(t)}\right)$*; so, we have*

$$\boldsymbol{x}^{(t)} = \operatorname{argmin} \lambda \|\boldsymbol{z}\|_1 + \frac{1}{2}\|\boldsymbol{z} - \boldsymbol{u}^{(t-1)}\|_2^2$$
$$= \operatorname{argmin} \sum_{n=1}^{N} \lambda \left|z_n\right| + \frac{1}{2}\left(z_n - u_n^{(t-1)}\right)^2$$

The objective function decouples in terms of n; thus,

$$x_n^{(t)} = \underset{z_n}{\operatorname{argmin}}\, \lambda \left|z_n\right| + \frac{1}{2}\left(z_n - u_n^{(t-1)}\right)^2$$

Iterative Soft Thresholding

By standard derivations, we can show

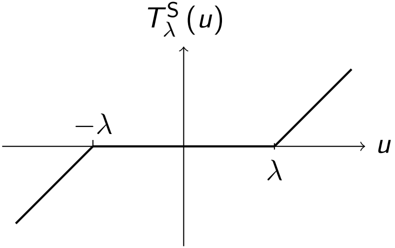
$$\operatorname*{argmin}_z \lambda |z| + \frac{1}{2}(z - u)^2 = \begin{cases} u - \lambda & u > \lambda \\ 0 & |u| \leq \lambda \\ u + \lambda & u < -\lambda \end{cases}$$

This is the s-called soft thresholding operator

Iterative Soft Thresholding

The soft thresholding operator is defined as

$$T_\lambda^{\mathsf{S}}(u) = \begin{cases} u - \lambda \mathrm{Sgn}(u) & |u| > \lambda \\ 0 & |u| \leq \lambda \end{cases}$$



Iterative Soft Thresholding: Derivation

So the recursive update rule reduces to

$$x_n^{(t)} = T_\lambda^{\mathrm{S}}\left(u_n^{(t-1)}\right)$$

Remember that

$$\begin{aligned}\boldsymbol{u}^{(t-1)} = \boldsymbol{u}\left(\boldsymbol{x}^{(t-1)}\right) &= \boldsymbol{x}^{(t-1)} + \beta\mathbf{A}^{\mathsf{T}}\boldsymbol{r}\left(\boldsymbol{x}^{(t-1)}\right)\\ &= \boldsymbol{x}^{(t-1)} + \beta\mathbf{A}^{\mathsf{T}}\left(\boldsymbol{y} - \mathbf{A}\boldsymbol{x}^{(t-1)}\right)\end{aligned}$$

Thus, the recursive update rule is written as

$$\boldsymbol{x}^{(t)} = T_\lambda^{\mathrm{S}}\left(\boldsymbol{x}^{(t-1)} + \beta\mathbf{A}^{\mathsf{T}}\left(\boldsymbol{y} - \mathbf{A}\boldsymbol{x}^{(t-1)}\right)\right)$$

Iterative Soft Thresholding

- *It starts with* $\boldsymbol{x}^{(0)} = \mathbf{0}$ *and* $\mathcal{S}^{(0)} = \emptyset$ *and a fixed* β
- *It updates the approximation of the signal as*

$$\boldsymbol{x}^{(t)} = T_{\lambda}^{\mathsf{S}}\left(\boldsymbol{x}^{(t-1)} + \beta \mathbf{A}^{\mathsf{T}}\left(\boldsymbol{y} - \mathbf{A}\boldsymbol{x}^{(t-1)}\right)\right)$$

- *It stops at iteration* T*, at which* $\mathbf{A}\boldsymbol{x}^{(T)} = \boldsymbol{y}$

Iterative Soft Thresholding

One can observe immediately that

Like OMP, this algorithm does not require s

What about performance compared to hard thresholding?

Soft thresholding shows better convergence

What about complexity?

Both hard and soft thresholding approaches are the same

Iterative Soft Thresholding

Attention

From basis pursuit denoising, one may assume that

λ *should be sent to zero!*

This is however not correct, since

Our iterative approach finds a sub-optimal solution

In practice,

λ *and* β *are tuned numerically*

Final Points

Summary

We learned greedy algorithms

- *Orthogonal Matching Pursuit*
- *Compressive Sampling Matching Pursuit*
- *Subspace Pursuit*

We also learned thresholding algorithms

- *Iterative Hard Thresholding*
- *Hard Thresholding Pursuit*
- *Iterative Soft Thresholding*

For each algorithm, we discussed pros and cons

What We Learn Next?

Up to now, we assumed noiseless sampling, i.e.,

Samples are modeled as $\mathbf{y} = \mathbf{A}\mathbf{x}$

However, in practice we usually have noisy samples

Samples are modeled as $\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{w}$

In the next part of the lecture we learn,

How to modify the algorithms, in order to cope with sampling noise

Which Parts of Textbooks?

We are over with this part

I would suggest to go over the textbook

A Mathematical Introduction to Compressive Sensing
S. Foucart and H. Rauhut, Book, 2013

and study the following parts:

- *Chapter 3: Sections 3.2 and 3.3*

Also a good paper discussing basic sparse recovery algorithms is

Maleki & Donoho "Optimally Tuned Iterative Reconstruction Algorithms for Compressed Sensing," in IEEE JSTSP, 2010

You can check it out here